



universidade
de aveiro

deti universidade de aveiro
departamento de eletrónica,
telecomunicações e informática

Relatório de Análise

Segurança Informática e nas Organizações
Departamento de Eletrónica, Telecomunicações e Informática

Ano Letivo 2023/2024
Projeto 2 ASVS - Equipa 21

André Miragaia – 108412
João Rodrigues – 108214
Guilherme Duarte – 107766
André Cruz – 110554
Guilherme Andrade – 107696

Índice

| | |
|---|----|
| Introdução..... | 3 |
| Password Strength Evaluation (V2.1)..... | 4 |
| Requisitos 2.1.1 / 2.1.2 | 4 |
| Requisitos 2.1.3 / 2.1.4 | 4 |
| Requisitos 2.1.5 / 2.1.6 | 6 |
| Requisito 2.1.7 | 7 |
| Requisito 2.1.8 | 8 |
| Requisito 2.1.9 | 9 |
| Requisitos 2.1.10 / 2.1.11 | 9 |
| Requisito 2.1.12 | 10 |
| Encrypted Database Storage (V6.1, V6.2, V8.3)..... | 11 |
| Data Classification – V6.1 (requisitos 6.1.1 / 6.1.3)..... | 11 |
| Algorithms – V6.2 (requisitos 6.2.2 / 6.2.3 / 6.2.4 / 6.2.5 / 6.2.6 / 6.2.7) | 14 |
| Sensitive Private Data – V8.3 (requisitos 8.3.1 / 8.3.2 / 8.3.3 / 8.3.4 / 8.3.5 / 8.3.7)..... | 15 |
| Requisito 14.3.1 | 16 |
| Análise | 16 |
| Procedimento..... | 16 |
| Requisito 3.3.2 | 17 |
| Análise | 17 |
| Procedimento..... | 17 |
| Requisitos 8.2.1 / 8.2.3 | 18 |
| Análise | 18 |
| Procedimento..... | 18 |
| Requisito 7.4.1 | 20 |
| Análise | 20 |
| Procedimento..... | 20 |
| Requisito 12.1.1 | 22 |
| Análise | 22 |
| Procedimento..... | 22 |

Introdução

Este documento apresenta uma análise detalhada das medidas de segurança implementadas relativamente à auditoria que realizamos à nossa aplicação tendo por base a ASVS-*checklist* que nos foi fornecida.

Foram implementadas 2 *software features*:

- Password strength evaluation;
- Encrypted database storage.

Foram feitas diversas melhorias e corrigidas falhas que vão de encontro aos seguintes requisitos de segurança:

- Requisito 3.3.2;
- Requisito 7.4.1;
- Requisito 8.2.1
- Requisito 8.2.3;
- Requisito 12.1.1;
- Requisito 14.3.1.

Dentro das *features* implementadas, foram validados os requisitos necessários para que a *feature* ficasse implementada de maneira correta. Esses requisitos estão especificados nas partes correspondentes a cada *feature*.

Password Strength Evaluation (V2.1)

Para a implementação desta *feature* implementámos os requisitos ASVS associados a Password Security Credentials (V2.1) que não estavam válidos na versão original (1^a entrega). Foi ainda feita a integração da haveibeenpwned API para verificar se a *password* introduzida foi *breached* (requisito 2.1.7).

Requisitos 2.1.1 / 2.1.2

Para que estes dois requisitos estejam válidos, é necessário que o tamanho da password seja de pelo menos 12 caracteres e não maior que 128. Na aplicação original verificava-se apenas se a password tinha um tamanho superior a 8 caracteres. Foi modificado com a função que se segue na figura.

```
function validatePassword($password){  
    $isLongEnough = strlen($password) >= 12;  
    $isNotTooLong = strlen($password) <= 128;  
    return $isLongEnough && $isNotTooLong;  
}
```

Figura 1: Função que verifica o tamanho da password (./api/functions.php)

Requisitos 2.1.3 / 2.1.4

Para que estes dois requisitos estejam válidos, é necessário que múltiplos espaços consecutivos sejam substituídos por um único espaço e que qualquer caracter *Unicode* seja incluído, e como podemos ver nas duas imagens da página de log-in e registo que se vão seguir, a nossa aplicação aceita quer espaços, quer emojis. A função que se segue faz a substituição dos espaços consecutivos por apenas um.

```
function processPassword($password) {  
    return preg_replace('/\s+/', ' ', $password);  
}
```

Figura 2: Função que verifica o tamanho da password (./api/functions.php)

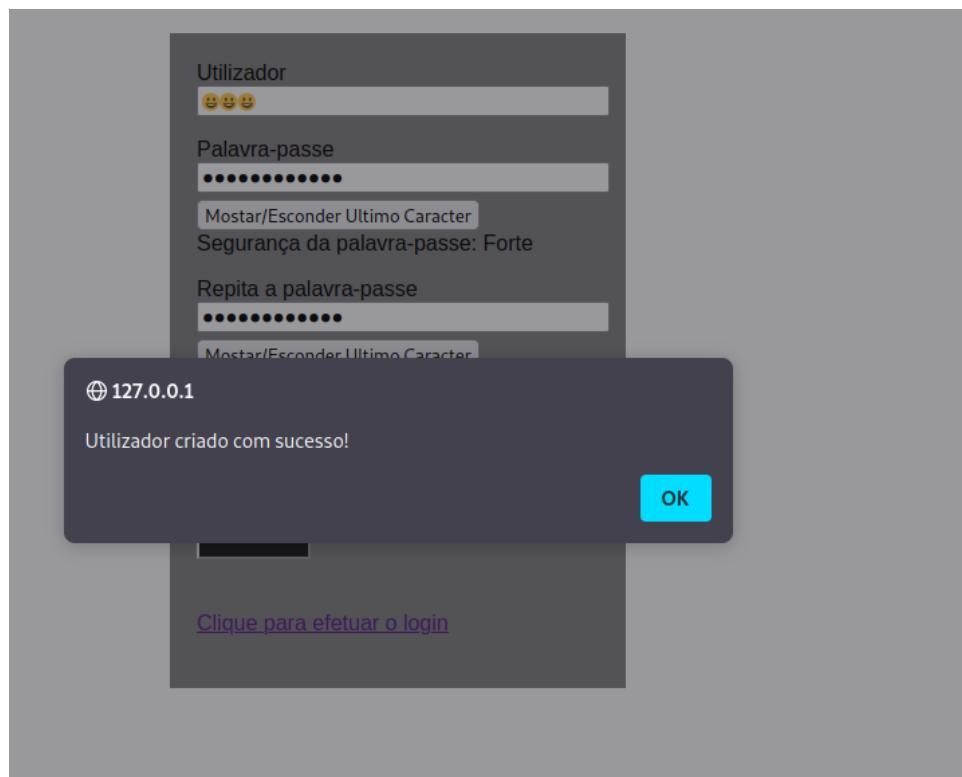


Figura 3: Registo feito com o user “😊😊😊” e password “😊😊😊😊😊😊😊😊”

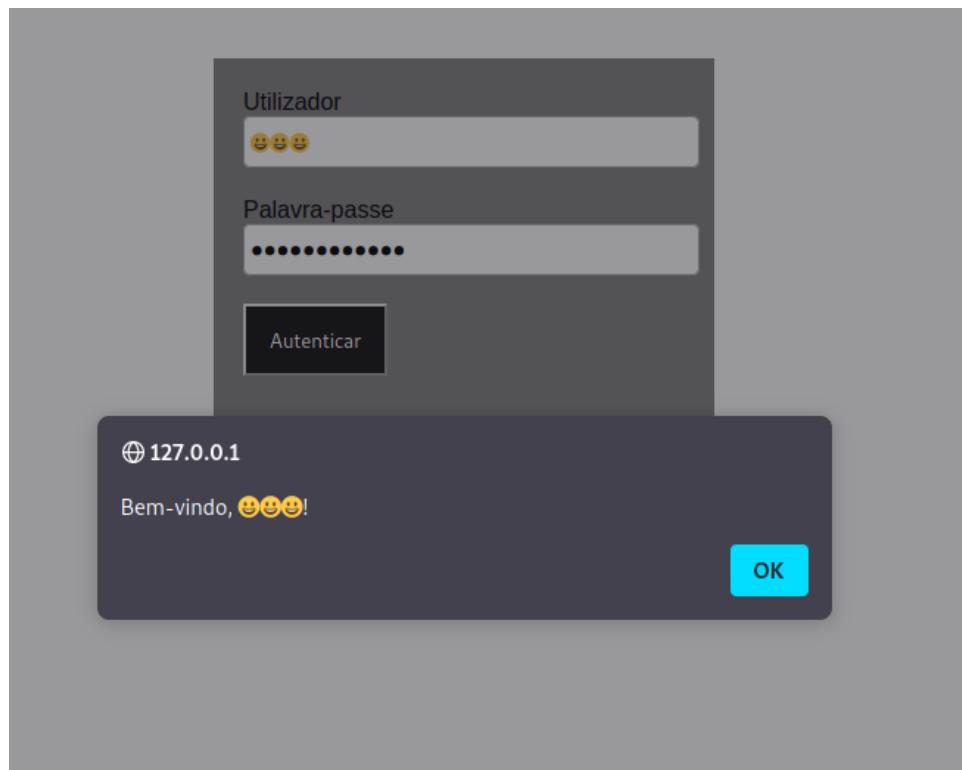


Figura 4 Log-in feito com o user “😊😊😊” e password “😊😊😊😊😊😊😊😊”

Requisitos 2.1.5 / 2.1.6

Para que estes dois requisitos estejam válidos, é necessário que haja a possibilidade de alterar a palavra-passe e que, para isso, o utilizador tenha que introduzir a palavra-passe atual. Estes dois requisitos já se encontravam válidos na aplicação original, pelo que não foi necessário fazer alterações.

Utilizador: julio2

Password: *****

Tipo de utilizador: Utilizador padrão

Alterar palavra-passe

Palavra-passe antiga:

Palavra-passe nova:

Repita a nova palavra-passe:

Strength:

Figura 5: Interface de alteração de password

```
if ($password == $hashedPassword) {  
    if($json["new_password"] == $json["new_password1"]){
        if( !(strlen($json["new_password"]) >= 8 && $uppercase && $number && $specialChars && $specialChars1)){
            echo json_encode(array("result" => 0, "result_text" => "A palavra-passe necessita de ter mais que 8 caracteres"));
            die();
        }
        $new_password = hash("sha256",$json["new_password"]);
        $sql = "UPDATE users set password= :new_password WHERE id = :id ";
        $updateStmt = $conn->prepare($sql);
        $updateStmt->bindParam(':id', $id, PDO::PARAM_INT);
        $updateStmt->bindParam(':new_password', $new_password, PDO::PARAM_STR);
        $updateStmt->execute();
        echo json_encode(array("result" => 1, "result_text" => "Palavra-passe alterada com sucesso!"));
    } else {
        echo json_encode(array("result" => 0, "result_text" => "Palavras-passe novas não são iguais"));
    }
} else {
    echo json_encode(array("result" => 0, "result_text" => "Palavra-passe não coincide com a atual"));
}
```

Figura 6: Verificação e comparação das passwords introduzidas (
./api/change_password.php)

Requisito 2.1.7

Para que este requisito esteja válido é necessário que exista uma comparação entre a *password* introduzida pelo utilizador e uma lista de *breached passwords*. Este requisito é importante para dificultar ataques *brute-force*, visto que o utilizador não poderá escolher uma *password* que esteja comprometida.

Para implementar esta funcionalidade, decidimos usar a haveibeenpwned API. Esta implementação basicamente compara uma *hash* da *password* introduzida com todas as *passwords* na base de dados da API (*breached passwords*), como consta na seguinte função.

```
function isPasswordBreached($password) {  
    $hashedPassword = strtoupper(hash('sha1', $password));  
    $prefix = substr($hashedPassword, 0, 5);  
    $suffix = substr($hashedPassword, 5);  
  
    $url = 'https://api.pwnedpasswords.com/range/' . $prefix;  
    $response = file_get_contents($url);  
    if ($response === false) {  
        return false;  
    }  
  
    $lines = explode("\n", $response);  
    foreach ($lines as $line) {  
        if (strpos($line, $suffix) !== false) {  
            return true; // Password is breached  
        }  
    }  
  
    return false; // Password is safe  
}
```

Figura 7: Verificação de palavras-passe comprometidas (./api/functions.php)

Realizando o teste com o utilizador “sporting” e *password* “sporting1906”, podemos verificar que não é registada a conta e é prontamente mostrado o alerta respetivo ao utilizador.

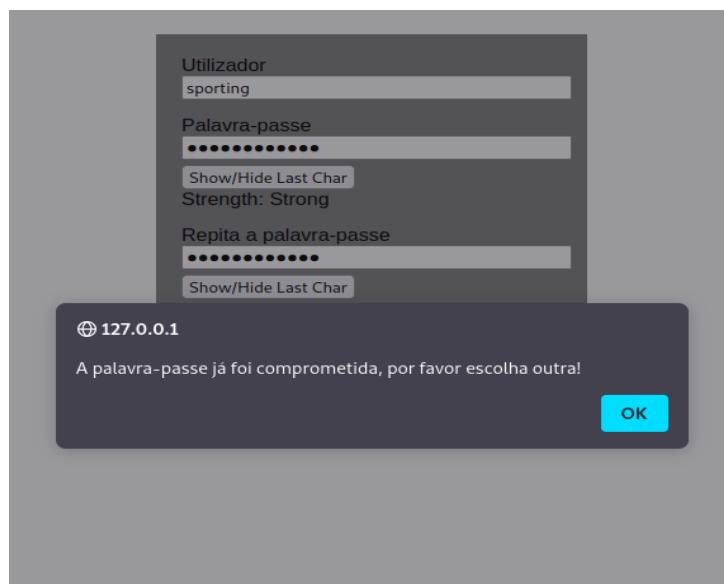


Figura 8: Alerta palavra-passe comprometida

Requisito 2.1.8

Para que este requisito esteja válido é necessário haver um medidor de força da *password* que indique quão forte é a *password*, à medida que esta é introduzida.

Esta medição é feita com base no princípio da entropia, cada tipo de caracteres (letras maiúsculas, minúsculas, números ou símbolos) tem um valor associado e este é somado aos valores dos tipos de caracteres existentes na *password*. O logaritmo deste valor resultante é multiplicado pelo tamanho da *password* de forma a obter o valor de entropia da mesma, que é utilizado para definir o seu nível de força.

```
function calculatePasswordStrength(password) {  
    var entropy = 0;  
    var poolSize = 0;  
  
    if (password.match(/[a-z]/)) poolSize += 26; // Lowercase letters  
    if (password.match(/[A-Z]/)) poolSize += 26; // Uppercase letters  
    if (password.match(/[0-9]/)) poolSize += 10; // Digits  
    if (password.match(/[\\W]/)) poolSize += 32; // Special characters  
  
    if (poolSize > 0) {  
        entropy = Math.log2(poolSize) * password.length;  
    }  
    var strengthLevel;  
    if (entropy < 28) {  
        strengthLevel = "Very Weak";  
    } else if (entropy < 36) {  
        strengthLevel = "Weak";  
    } else if (entropy < 60) {  
        strengthLevel = "Moderate";  
    } else if (entropy < 128) {  
        strengthLevel = "Strong";  
    } else {  
        strengthLevel = "Very Strong";  
    }  
  
    return strengthLevel;  
}
```

Figura 9: Função para calcular a força da *password* (./api/functions.php)

Os níveis de força definidos foram “Very Weak”, “Weak”, “Moderate”, “Strong” e “Very Strong”.

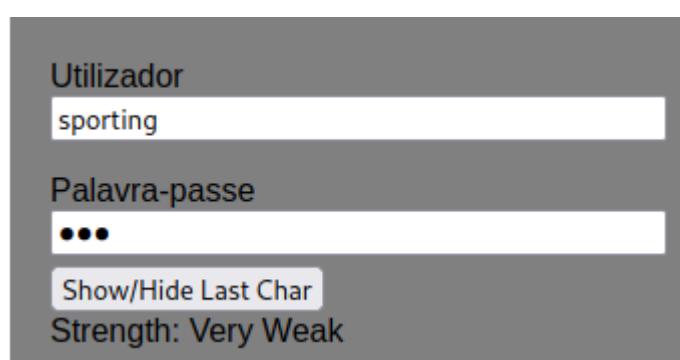


Figura 10: Palavra-passe com nível de força “Very Weak”

| | |
|----------------------------|----------|
| Utilizador | sporting |
| Palavra-passe | •••••• |
| Show/Hide Last Char | |
| Strength: Moderate | |

Figura 11: Palavra-passe com nível de força "Moderate"

| | |
|------------------------------|----------------------|
| Utilizador | sporting |
| Palavra-passe | •••••••••••••••••••• |
| Show/Hide Last Char | |
| Strength: Very Strong | |

Figura 12: Palavra-passe com nível de força "Very Strong"

Requisito 2.1.9

Para que este requisito esteja válido a aplicação não pode ter regras relativas ao tipo de caracteres que compõe a palavra-passe. Na aplicação original era feita a verificação da existência de maiúsculas, números e símbolos (tinha de existir pelo menos 1 de cada). De forma a tornar este requisito válido, esta verificação foi removida, passando a existir apenas a obrigação do tamanho ser inferior a 128 e maior que 12, como já visto anteriormente.

Requisitos 2.1.10 / 2.1.11

Para que estes dois requisitos estejam válidos não pode existir a necessidade de rotação periódica de credenciais ou requisitos a nível do histórico de *passwords*. Para além disso tem de ser possível fazer *paste* da *password* para que não haja problemas com gestores de *passwords* do browser ou externos.

Ambos os requisitos já se encontravam válidos na aplicação original pelo que não foram efetuadas alterações a estas funcionalidades.

Requisito 2.1.12

Para que este requisito esteja válido é necessário que o utilizador consiga ver temporariamente toda a password introduzida ou o último caracter introduzido. Optámos por implementar uma função para ver apenas o último caracter introduzido, visto que é mais seguro especialmente em locais públicos ou com pessoas à volta.

```
function toggleLastCharVisibility(passwordFieldId) {
    var input = document.getElementById(passwordFieldId);
    var isPassword = input.type === 'password';

    if (isPassword && input.value.length > 0) {
        input.type = 'text';
        var originalValue = input.value;
        input.value = '*'.repeat(input.value.length - 1) + originalValue.substr(-1);
        setTimeout(() => {
            input.type = 'password';
            input.value = originalValue;
        }, 1000);
    }
}
```

Figura 13: Função para ver o último caracter inserido (./api/functions.php)

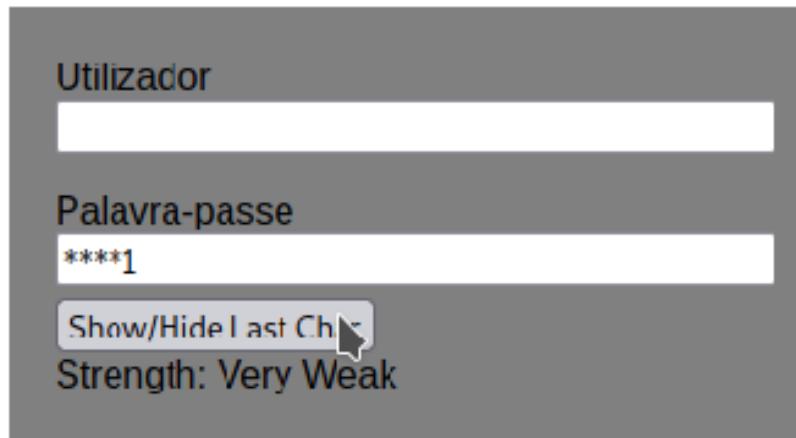


Figura 14: Ver o último caracter inserido

Encrypted Database Storage (V6.1, V6.2, V8.3)

Para a implementação desta funcionalidade, foram incorporados requisitos da ASVS relacionados ao Encrypted Database Storage (V6.1, V6.2, V8.3) que não estavam válidos na versão original (1^a Entrega).

O propósito principal desta adição é assegurar a segurança e integridade dos dados, alinhados a uma política transparente de tratamento de informações para os utilizadores.

Data Classification – V6.1 (requisitos 6.1.1 / 6.1.3)

O foco principal no ASVS 6.1 é a confidencialidade das informações armazenadas, assegurando que os dados sensíveis dos usuários permaneçam cifrados enquanto estão em repouso. Para atender a esse requisito, optamos por duas abordagens:

- Cifrar todos os dados com o MariaDB;
- Cifrar os dados sensíveis com AES-256-CBC.

Na primeira abordagem usámos uma funcionalidade presente no MariaDB em conjunto com o plugin “file_key_management” para garantir a confidencialidade de todos os dados presentes na base de dados. Para tal adicionámos as seguintes opções ao arquivo de configuração MariaDB.cnf (configs/mariadb.cnf):

- **plugin_dir** – Pasta onde estão presentes os plugins;
- **plugin_load_add** – Plugin que queremos usar (neste caso o file_key_management);
- **file_key_management_filename** – Arquivo onde estão as chaves;
- **file_key_management_filekey** – Arquivo que tem a password para decifrar as chaves;
- **file_key_management_encryption_algorithm** – Algoritmo que queremos usar;
- **innodb_encrypt_tables** – Permite-nos escolher se queremos cifrar as tabelas innodb;
- **innodb_encrypt_temporary_tables** – Permite-nos escolher se queremos criptografar as tabelas temporárias innodb;
- **innodb_encrypt_log** – Permite-nos escolher se queremos cifrar os logs innodb;
- **innodb_encryption_threads** – Permite-nos escolher o número de unidades de processamento que depositamos quando ciframos as tabelas innodb;
- **innodb_encryption_rotate_key_age** – Permite-nos escolher de quanto em quanto tempo as chaves serão renovadas;
- **encrypt-tmp-disk-tables** – Permite-nos escolher se queremos cifrar as tabelas temporárias presentes no disco;
- **encrypt-tmp-files** – Permite-nos escolher se queremos cifrar os arquivos temporários;
- **encrypt-binlog** – Permite-nos escolher se queremos cifrar os arquivos de log binários;
- **aria_encrypt_tables** – Permite-nos escolher se queremos cifrar as tabelas aria.

```

[mysqld]
# at-rest encryption
plugin_dir=/usr/lib/mysql/plugin
plugin_load_add=file_key_management.so
file_key_management_filename = /etc/mysql/rest/keyfile.enc
file_key_management_filekey = FILE:/etc/mysql/rest/keyfile.key
file_key_management_encryption_algorithm = AES_128_CBC

innodb_encrypt_tables = ON      • Criptografar todos os dados através de um
innodb_encrypt_temporary_tables = ON      • Criptografar os dados sensíveis com AES
innodb_encrypt_log = ON
innodb_encryption_threads = 4      Começando por falar da primeira abordagem
innodb_encryption_rotate_key_age = 1      MariaDB junto com o plugin file_key_management
encrypt-tmp-disk-tables = 1      presentes na base de dados estão inacessíveis ex
encrypt-tmp-files = 1      configuração do MariaDB.cnf
encrypt-binlog = 1
aria_encrypt_tables = ON

```

Figura 15: Opções adicionadas

Através desta configuração conseguimos garantir a segurança da nossa base de dados. No entanto, é necessário gerar as chaves e a *password* e para tal criamos o seguinte script em bash que nos automatizou o processo:

```

#!/bin/bash

echo "Digite quantas chaves pretende gerar:"
read userInput

if [[ $userInput =~ ^[1-9][0-9]*$ ]]; then
    intValue=$((userInput))
    mkdir -p /etc/mysql/rest
    # Este for gera as chaves que queremos e armazena no arquivo /etc/mysql/rest/keyfile
    for i in $(seq 1 $intValue); do
        echo -n "$i;" ; openssl rand -hex 32
    done | sudo tee -a /etc/mysql/rest/keyfile
    # Gera uma password de 128 caracteres que é armazenado /etc/mysql/rest/keyfile.key
    sudo openssl rand -hex 128 | sudo tee -a /etc/mysql/rest/keyfile.key
    # Gera um arquivo criptografado em /etc/mysql/rest/keyfile.enc com as chaves
    # guardadas em /etc/mysql/rest/keyfile
    # através da password armazenada em file:/etc/mysql/rest/keyfile.key
    openssl enc -aes-256-cbc -md sha1 -pass file:/etc/mysql/rest/keyfile.key -in
    /etc/mysql/rest/keyfile -out /etc/mysql/rest/keyfile.enc
    # Apaga o arquivo não criptografado com as chaves
    rm /etc/mysql/rest/keyfile
    chown mysql:mysql -R /etc/mysql/rest
    chmod 600 /etc/mysql/rest/*
    echo "Exit ... "
else
    echo "Por favor, insira um número inteiro positivo!\nExit ... "
fi

```

Figura 16: Script para gerar as chaves

Através deste script presente em configs/key_generator.sh conseguimos gerar as chaves que queremos e assim podemos garantir que todos os dados presentes na base de dados estão cifrados corretamente.

Na segunda abordagem decidimos cifrar os dados sensíveis outra vez dando uma segunda camada de criptografia, mas desta vez usando as funções implícitas no PHP que se seguem.

```
$master_password = "0aaffea3cf7a98dd4e2188d89d652f96206fd8ab15b381dfd7f946293f28e339075d  
a0f10e52f3f1c0ca96dd2d23b0d2a08ae21bcc744a32bb5149bfa59fd830";  
  
function saltGenerator(){  
    return bin2hex(random_bytes(128));  
}  
  
function keyGenerator($key, $salt){  
    return hash("sha512", $key . $salt);  
}  
  
function encryptData($plaintext, $key){  
    $method = 'aes-256-cbc';  
    $iv = openssl_random_pseudo_bytes(openssl_cipher_iv_length($method));  
    $ciphertext = openssl_encrypt($plaintext, $method, $key, OPENSSL_RAW_DATA, $iv);  
    $hmac = hash_hmac('sha256', $ciphertext, $key, true);  
    $encryptedData = base64_encode($iv . $hmac . $ciphertext);  
    return $encryptedData;  
}
```

Figura 17: Funções presentes no /functions.php

A função encryptData retorna uma *string* em base64 onde estão inclusos o vetor de inicialização, o HMAC e o texto cifrado.

Para garantir segurança nós seguimos o seguinte procedimento:

- **\$key** – A nossa *password* que é a junção da \$master_password + \$salt que é gerado pelo saltGenerator, implementado em ./api/checkout.php;
- **\$plaintext** – O texto que queremos cifrar ao qual ainda é somado o \$salt, implementado em ./api/checkout.php;
- **\$iv** – Gerado de forma aleatória para garantir uma maior segurança;
- **\$ciphertext** – Os dados são cifrados com AES-256-CBC onde usamos o OPENSSL_RAW_DATA como *padding*;
- **\$hmac** – Usado para criar uma chave em SHA256 para garantir que não houve manipulação de dados pois, caso haja, a chave gerada ao decifrar não irá coincidir com a chave gerada quando os dados foram cifrados;
- **\$encryptedData** – Soma o vetor de inicialização, o HMAC e os dados cifrados codificando-os depois em base64.

Assim, através desta abordagem podemos garantir que os dados estão devidamente cifrados de forma a garantir que os dados sensíveis dos utilizadores não são acessados ou alterados por pessoas mal-intencionadas.

Algorithms – V6.2 (requisitos 6.2.2 / 6.2.3 / 6.2.4 / 6.2.5 / 6.2.6 / 6.2.7)

O foco principal no ASVS 6.2 é se os algoritmos criptográficos usados e a forma como foram implementados são recomendados pelo ramo ou pelo governo e se as informações estão de facto cifradas.

Como foi mencionado anteriormente, as regras usadas para cifrar os dados confidenciais na segunda abordagem utilizam:

- Um vetor de inicialização único para cada situação visto que o mesmo é gerado de forma aleatória;
- Uma *password* única para cada situação pois a *password* é a soma da \$master_password com o \$salt gerado pelo saltGenerator;
- O uso da criptografia AES-256-CBC que é amplamente reconhecida no ramo pela sua segurança;
- Uso de HMAC para garantir a integridade dos dados de forma a evitar que estes sejam alterados e, caso aconteça, o HMAC gerado quando se tenta decifrar os dados não irá coincidir com o HMAC armazenado;
- O *padding* é feito através do OPENSSL_RAW_DATA para gerar valores aleatórios.

No entanto esta abordagem não permite uma troca de chaves o que pode prejudicar em parte a segurança.

Para solucionar o problema decidimos que a primeira abordagem deveria preencher as lacunas da segunda e como foi mencionado anteriormente, no arquivo de configuração do MariaDB implementámos a opção “innodb_encryption_rotate_key_age = 1” e assim conseguimos garantir que quando a chave usada atinge um ciclo de uso, a chave é trocada por outra mantendo a segurança dos dados.

Por último, falta apenas descrever que algoritmo foi usado para as passwords e inicialmente o algoritmo utilizado era o SHA-256. No entanto, chegamos à conclusão de que existem algoritmos mais seguros e por essa razão decidimos usar o Argon2 que garantidamente é muito superior ao algoritmo anterior. O algoritmo foi implementado em api/signup.php e para explicar a razão pela qual este algoritmo é muito superior ao anterior vamos analisar o código correspondente:

```
$hash = password_hash($password, PASSWORD_ARGON2ID, ['memory_cost' => 1<<12, 'time_cost' => 12, 'threads' => 6]);
```

Como podemos ver na instrução o Argon2 baseia-se em três argumentos principais que são:

- **memory_cost** – Quantidade de memória usada para gerar a *hash*;
- **time_cost** – Tempo depositado para gerar a *hash*;
- **threads** – Número de unidades de processamento usadas para gerar a *hash*.

Com isto a conclusão factual que podemos obter é que este algoritmo dificulta bastante os ataques de força bruta e assim podemos garantir que as *passwords* estão de facto seguras. Portanto, podemos concluir que foram cumpridas as metas relativas ao ASVS 6.2.

Sensitive Private Data – V8.3 (requisitos 8.3.1 / 8.3.2 / 8.3.3 / 8.3.4 / 8.3.5 / 8.3.7)

O foco principal no ASVS 8.3 é sobre a política de tratamento de dados, a transparência com o utilizador sobre como os dados são tratados e o fornecimento de mecanismos que dão mais controlo ao utilizador sobre as suas informações.

Para tal foi criada uma política de dados que o utilizador é obrigado a aceitar caso se queira registar. Nestes termos, a linguagem é clara e coesa sobre como os dados são usados.

Política de Privacidade e Consentimento de Dados do Departamento de Eletrónica Telecomunicações e Informática (DETI)

Que dados coletamos?

Todos os dados que coletamos são:
* Dados de registo;
* Dados de pagamento.

De que forma usamos os dados?

Os dados são usados apenas com o objetivo de que a plataforma consiga realizar os serviços solicitados.

Alguma entidade pode ter acesso aos dados?

Não, excepto por questões judiciais caso solicitado.
No que toda a pessoas mal intencionadas nós somos muito sérios quanto a isso e por isso que os dados estão devidamente criptografados com as melhores criptografia do mercado.

Posso apagar os meus dados quando necessário?

Sim! Nós não só disponibilizamos a opção de apagar os dados com também fornecemos a opção de os baixar caso pretenda.

Figura 18: Política de Privacidade

Como é possível visualizar nos termos, no último ponto foi mencionado que o utilizador pode fazer download dos dados ou até mesmo apagá-los. Esta opção foi implementada no profile.php, para garantir que o utilizador possa ter maior controlo sobre os seus dados.

Utilizador: antonio

Password: *****

Tipo de utilizador: Utilizador padrão

Alterar palavra-passe

Palavra-passe antiga

Palavra-passe nova

Repita a nova palavra-passe

Mudar palavra-passe

Baixar dados da conta

Apagar a conta

Figura 19: Opção de fazer download ou eliminar os dados da conta

Como já discutido nos tópicos anteriores, toda a informação sensível está devidamente cifrada para garantir o sigilo da mesma, fazendo-se acompanhar de uma política de dados transparente com o utilizador. No entanto, as políticas adotadas neste caso não são muito exigentes visto que os dados coletados são o utilizador, a password e os produtos comprados.

Requisito 14.3.1

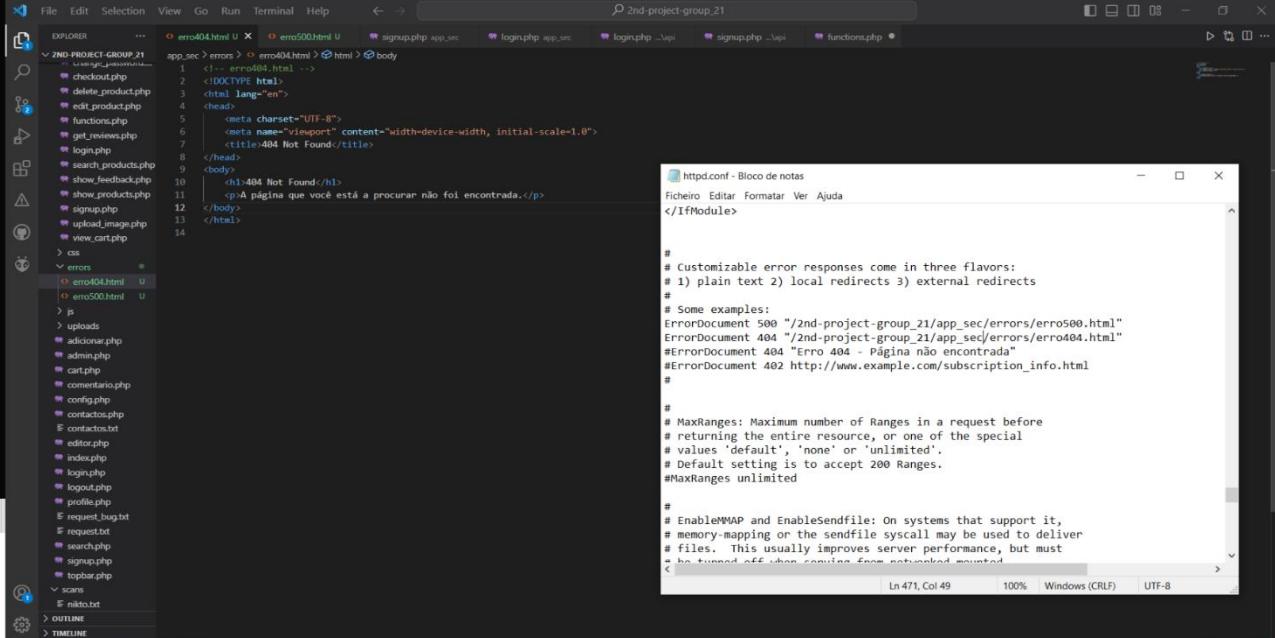
Análise

Decidimos resolver esta vulnerabilidade pois, realizámos uma pesquisa sobre ela e chegámos à conclusão que se pode tornar perigosa para a segurança da nossa aplicação, dado que erros detalhados podem conter informações sensíveis sobre a infraestrutura, o código fonte ou até mesmo credenciais do sistema. Ao fornecer mensagens genéricas e amigáveis, evitamos expor inadvertidamente detalhes internos do sistema, reduzindo assim potenciais riscos de segurança e criando uma experiência mais positiva para o utilizador final.

Procedimento

As ações realizadas para tornar este requisito, que podia comprometer a segurança, válido, foram a criação de duas páginas HTML para os erros 404 e 500 e a alteração das configurações do apache para mostrar as nossas mensagens de erro em vez das que eram mostradas anteriormente pelo servidor.

Exemplo código que foi realizado e alterado:



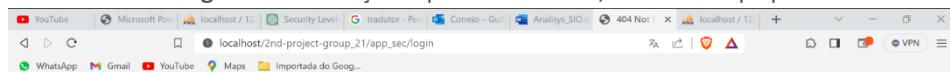
```
functions.php
1 <?php
2 // -> erro404.html -->
3 <!DOCTYPE html>
4 <html lang="en">
5   <head>
6     <meta charset="UTF-8">
7     <meta name="viewport" content="width=device-width, initial-scale=1.0">
8     <title>404 Not Found</title>
9   </head>
10  <body>
11    <h1>404 Not Found</h1>
12    <p>A página que você está a procurar não foi encontrada.</p>
13  </body>
14 </html>
```

```
httpd.conf - Bloco de notas
Ficheiro Editar Formatar Ver Ajuda
</IfModule>

#
# Customizable error responses come in three flavors:
# 1) plain text 2) local redirects 3) external redirects
#
# Some examples:
ErrorDocument 500 "/2nd-project-group_21/app_sec/errors/erro500.html"
ErrorDocument 404 "/2nd-project-group_21/app_sec/errors/erro404.html"
ErrorDocument 404 "Erro 404 - Página não encontrada"
ErrorDocument 402 http://www.example.com/subscription_info.html
#
#
# MaxRanges: Maximum number of Ranges in a request before
# returning the entire resource, or one of the special
# values 'default', 'none' or 'unlimited'.
# Default setting is to accept 200 Ranges.
MaxRanges unlimited

#
# EnableMMAP and EnableSendfile: On systems that support it,
# memory-mapping or the sendfile syscall may be used to deliver
# files. This usually improves server performance, but must
# be turned off when serving from networked-mounted
C
```

Figura 20: Funções presentes no /functions.php



404 Not Found

A página que você está a procurar não foi encontrada.

Figura 21: Página mostrada com erro 404

Requisito 3.3.2

Análise

Optámos por tornar este requisito válido, pois é interessante para aumentar a segurança da nossa aplicação. Requerer reautenticação regularmente reduz o risco de acesso não autorizado, especialmente se um utilizador deixar a sua sessão aberta num dispositivo partilhado, ajudando assim a prevenir ataques de sessão.

Procedimento

Para validar este requisito, decidimos criar uma função em PHP em que primeiro definimos o tempo máximo de inatividade antes da reautenticação, de seguida definimos o tempo máximo de uso ativo sem fazer a reautenticação e depois calculamos o tempo decorrido desde a última atividade na sessão.

Se o tempo da sessão for maior que o tempo de inatividade e o utilizador estiver com sessão iniciada, esta é terminada. O mesmo acontece se a sessão estiver ativa durante 12 horas.

```
function needs_reauthentication(){
    $idle_timeout = 1800; // 30 minutos de inatividade antes da reautenticação
    $active_timeout = 43200; // 12 horas de uso ativo antes da reautenticação

    $session_time = time() - $_SESSION['timeout'];

    if ($session_time > $idle_timeout && check_login_boolean()) {
        exit_session();
        exit();
    } elseif ($session_time > $active_timeout && check_login_boolean()) {
        exit_session();
        exit();
    }
}
```

Figura 22: Função de reautenticação (./api/functions.php)

Requisitos 8.2.1 / 8.2.3

Análise

Optámos por validar o requisito 8.2.1 que está relacionado com a configuração de cabeçalhos *anti-caching*, visto que, a presença de dados sensíveis em caches de *browsers* pode levar a exposições indesejadas de informação. Isto seria crítico para a nossa aplicação por se tratar de uma loja *online* que lida com dados pessoais, financeiros ou outros tipos de informação confidencial.

Já o requisito 8.2.3 está relacionado com a verificação da *storage* do cliente, no término da sessão, assegurando que esta fica limpa. Existe aqui um risco visto que podem ser deixadas informações residuais que permitam que um atacante faça reutilização de *tokens* de sessão válidos ou de outros dados autenticados. Assegurando que a *storage* do cliente seja limpa, minimiza-se ainda a exposição a ataque baseados em *scripts*, como *Cross-Site Scripting (XSS)* e *Cross-Site Request Forgery (CSRF)*.

Procedimento

Para validar o requisito 8.2.1 optámos pela configuração de cabeçalhos como vemos nas figuras seguintes, estes foram aplicados às páginas mais sensíveis onde ocorrem atualizações/mudanças na aplicação.

```
header("Cache-Control: no-store, no-cache,  
must-revalidate, max-age=0");  
header("Pragma: no-cache");  
header("Expires: 0");
```

Figura 23: Headers aplicados nas páginas mais sensíveis

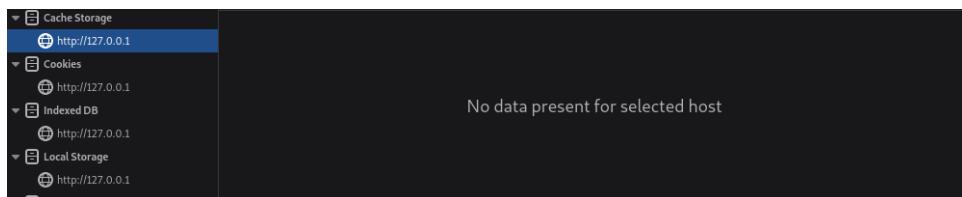


Figura 24: Cache Storage limpo após iteração pela aplicação

Para validar o requisito 8.2.3 foram feitas algumas adições de modo a conseguirmos uma melhoria no que diz respeito ao armazenamento inseguro de informações sensíveis.

Adicionámos a função que se segue, que faz a atribuição dos parâmetros dos *cookies* da sessão, as flags "secure" e "httponly" foram configuradas de modo a que os *cookies* sejam transmitidos apenas através de conexões HTTPS e não sejam acessíveis via JavaScript, o que acrescenta alguma robustez. A variável "samesite" recebeu o valor "Lax" o que assegura que os *cookies* não sejam enviados através de pedidos de origens cruzadas (*Cross-Origin Requests*), prevenindo ataques de CSRF.

```

function setupSession() {
    $cookieParams = session_get_cookie_params();
    session_set_cookie_params([
        'lifetime' => $cookieParams["lifetime"],
        'path' => '/',
        'domain' => $cookieParams["domain"],
        'secure' => true,
        'httponly' => true,
        'samesite' => 'Lax'
    ]);
    session_start();
}

```

Figura 25: Função de *setup* à sessão (./api/functions.php)



Figura 26: Cookies quando uma sessão é criada

Foi também adicionada uma página intermédia para quando for executado o *logout*, se ter a certeza de que foram removidos os dados sensíveis da sessão e do *storage local*, garantimos assim que informações residuais não permaneçam no navegador do utilizador após a conclusão da sua sessão.

```

<!DOCTYPE html>
<html>
<head>
    <title>Logging Out...</title>
    <script>
        localStorage.clear();
        sessionStorage.clear();
        window.location.href = 'login.php';
        console.log("Logged out");
    </script>
</head>
<body>
    Logging out, please wait...
</body>
</html>

```

Figura 27: Página de *logout* que assegura a limpeza do *storage local*

Requisito 7.4.1

Análise

Optámos por validar este requisito relacionado à exibição de uma mensagem genérica em casos de erros inesperados ou sensíveis à segurança, possivelmente acompanhada de uma identificação única para facilitar a investigação por parte do pessoal de suporte.

Esta decisão foi fundamentada na importância de fornecer uma resposta transparente e informativa aos usuários em situações de erro, garantindo ao mesmo tempo a segurança da aplicação.

A implementação de uma mensagem genérica, juntamente com uma identificação única, proporciona aos utilizadores uma compreensão mais clara do ocorrido, ao mesmo tempo em que permite ao suporte técnico realizar uma investigação eficiente e precisa. A inclusão da identificação única é particularmente valiosa, pois facilita a correlação entre relatórios de usuários e registo de erro, simplificando o processo de resolução de problemas.

Procedimento

Na versão original já eram mostradas algumas mensagens de erro, mas decidimos reformular a forma como estávamos a lidar com erros conhecidos. Assim implementámos exceções que geram um “unique_id” para cada erro.

```
try {
    $data = file_get_contents('php://input');
    $json = json_decode($data, true);

    // Check if user is logged in
    if (!check_login_boolean()) {
        throw new Exception("Sessão não iniciada!");
    } elseif (!$json != NULL && key_exists("id", $json) && key_exists("comentario", $json)) {
        throw new Exception("JSON inválido!");
    } elseif (!key_exists("stars", $json)) {
        throw new Exception("Não foi adicionada nenhuma avaliação! Comentário não inserido!");
    } elseif (empty($json["id"]) || empty($json["stars"]) || empty($json["comentario"])) {
        throw new Exception("Um dos campos está vazio! Comentário não inserido!");
    } elseif (!is_numeric($json["id"]) && (int)$json["id"] > 0) {
        throw new Exception("O ID é inválido! Comentário não inserido!");
    } elseif (!is_numeric($json["stars"]) && (int)$json["stars"] >= 0 && (int)$json["stars"] <= 5) {
        throw new Exception("Número de estrelas inválido! Comentário não inserido!");
    } else {
        $id = (int)$json["id"];

        // Check if the product exists
        $query = "SELECT * FROM products WHERE id=:id LIMIT 1";
        $stmt = $conn->prepare($query);
        $stmt->bindParam(":id", $id, PDO::PARAM_INT);
        $stmt->execute();
        $row = $stmt->fetch();

        if (!$row) {
            throw new Exception("Produto não encontrado na base de dados! Comentário não inserido!");
        }

        $stars = $json["stars"];
        $comentario = htmlspecialchars($json["comentario"]);
        $username = $_SESSION["username"];

        // Insert comment into the database
        $sql = "INSERT INTO comments (id, username, stars, comment) VALUES (:id, :username, :stars, :comentario)";
        $stmt = $conn->prepare($sql);
        $stmt->bindParam(":id", $id, PDO::PARAM_INT);
        $stmt->bindParam(":username", $username, PDO::PARAM_STR);
        $stmt->bindParam(":stars", $stars, PDO::PARAM_STR);
        $stmt->bindParam(":comentario", $comentario, PDO::PARAM_STR);
        $stmt->execute();

        echo json_encode(array("result" => 1, "result_text" => "Comentário adicionado com sucesso!"));
    }
} catch (Exception $e) {
    $errorId = uniqid();
    error_log("Error ID: $errorId - " . $e->getMessage());
    $message = $e->getMessage();
    echo json_encode(array("result" => 0, "result_text" => $message . " Por favor, entre em contato com o suporte e forneça o ID do erro: $errorId"));
}
```

Figura 28: Exemplo da implementação de exceções (.api/add_feedback.php)

localhost diz

Imagen não foi adicionada com sucesso! O arquivo é muito grande.
Por favor, entre em contato com o suporte e forneça o ID do erro:
658f0ec50ad1a

OK

Figura 29: Exemplo de mensagem de erro apresentada (upload de uma imagem)

Requisito 12.1.1

Análise

A decisão de validar este requisito que está relacionado à restrição do *upload* de arquivos grandes foi motivada por uma série de considerações críticas para a integridade e desempenho da aplicação. Ao verificar e assegurar que a aplicação não aceite arquivos de grande porte, mitigamos riscos substanciais, incluindo a possibilidade de esgotamento de armazenamento e a potencial ocorrência de ataques de negação de serviço (DoS).

O limite no tamanho dos arquivos é crucial para prevenir a saturação do armazenamento disponível na infraestrutura da aplicação e pode-se preservar assim a integridade do armazenamento e a disponibilidade dos recursos essenciais.

Procedimento

Para resolver este requisito colocámos um limite de 5 MB para todas as imagens que forem *uploaded*.

```
$maxFileSize = 5 * 1024 * 1024; // 5MB
if ($_FILES['image']['size'] > $maxFileSize) {
    throw new Exception("Imagem não foi adicionada com sucesso! O arquivo é muito grande.");
}
```

Figura 29: Implementação do limite de 5MB para imagens